Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 55 (2008) 2005–2011
An International Journal
computers & mathematics with applications
www.elsevier.com/locate/camwa

Design of new XOR-based hash functions for cache memories☆

Sung-Jin Cho[a,*], Un-Sook Choi[b], Yoon-Hee Hwang[c], Han-Doo Kim[d]

[a] *Division of Mathematical Sciences, Pukyong National University, Busan 608-737, Republic of Korea*
[b] *Department of Multimedia Engineering, Tongmyong University, Busan 608-711, Republic of Korea*
[c] *Department of Information Security, Graduate School, Pukyong National University, Busan 608-737, Republic of Korea*
[d] *School of Computer Aided Science, Inje University, Gimhae, 621-749, Republic of Korea*

Received 12 April 2006; accepted 27 July 2007

Abstract

A hash function H is a computationally efficient function that maps bitstrings of arbitrary length to bitstrings of fixed length, called hash values. Hash functions have a variety of general computational uses. They are used in processors to augment the bandwidth of an interleaved multibank memory or to enhance the utilization of a prediction table or a cache. In this paper, we design new XOR-based hash functions, which compute each set index bit as XOR of a subset of the bits in the address by using the concepts of rank and null space. These are conflict-free hash functions which are of different types according to whether m is even or odd. To apply the constructed hash functions to the skewed-associative cache, we show that the degree of interbank dispersion between two hash functions is maximal.

Keywords: XOR-based hash functions; Conflict-free; 2-way skewed-associative caches; Interbank dispersion; Null space; Rank

1. Introduction

A hash function H is a computationally efficient function that maps bitstrings of arbitrary length to bitstrings of fixed length, called hash values. Hash functions have a variety of general computational uses. They are used in processors to augment the bandwidth of an interleaved multibank memory or to enhance the utilization of a prediction table or a cache [1–6]. The basic requirements for designing hash function are: the input can be of any length, the output has a fixed length, $H(x)$ is relatively easy to compute for any given x, $H(x)$ is 1-way, and $H(x)$ is conflict-free. Bank conflicts can severely reduce the bandwidth of an interleaved multibank memory and conflict misses increase the miss rate of a cache. Therefore it is important that a hash function has to succeed in spreading the most frequently occurring patterns over all indices. So it maps the elements of the frequently occurring patterns without conflicts.

Frailong et al. [7] suggested XOR-schemes that allowed parallel access to all previously mentioned data templates, which are of use in image processing and numerical analysis. Vandierendonck et al. [8] constructed XOR-based

☆ This work was supported by grant No. (R01-2006-000-10260-0) from the Basic Research Program of the Korea Science and Engineering Foundation.
* Corresponding author.
E-mail addresses: sjcho@pknu.ac.kr (S.-J. Cho), choies@tu.ac.kr (U.-S. Choi), yhhwang@pknu.ac.kr (Y.-H. Hwang), mathkhd@inje.ac.kr (H.-D. Kim).

0898-1221/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2007.07.008

hash functions which provided conflict-free mapping for a number of patterns for multibank memories and skewed-associative caches. They used fundamental notions of null space and column space for constructing these functions [8]. Their functions map $2m$ bits to $m(= 2k)$ bits which are conflict-free. But they did not construct hash functions which are conflict-free when m is odd. Also they constructed two XOR-based hash functions for skewed-associative caches [8–10]. But the degree of interbank dispersion between two hash functions is less than $2m$, which is the maximum degree between them. So they changed the basis of one hash function to overcome this problem.

In this paper, we design new XOR-based hash functions, which compute each set index bit as XOR of a subset of the bits in the address by using the concepts of rank and null space [11–14]. These are conflict-free hash functions which are of different types according to whether m is even or odd. To apply the constructed hash functions to the skewed-associative cache, we show that the degree of interbank dispersion between two hash functions is maximal. And thus we need not change the basis of the null space of the constructed hash functions.

2. Hash functions

In this section we introduce XOR-based hash functions in a general way and then we describe their representation by a vector–matrix product. Hash functions are used in processors to increase the bandwidth of an interleaved multibank memory or to improve the use of a cache. They map an address to a set index. Since all indices are used, the function should be surjective.

Definition 2.1. A hash function is a function from $\{0, \ldots, 2^n - 1\}(:= \mathbf{A})$ of n bit addresses to $\{0, \ldots, 2^m - 1\}(:= \mathbf{S})$ of m bit indices $(m < n)$.

Especially XOR-based hash functions can be modeled by means of a matrix representation and by using null spaces.

1. The matrix representation

An n bit address $\mathbf{a}$ is represented by a bit vector $a_1, \ldots, a_n$. A hash function mapping n bits to m bits is represented as a binary matrix H with n rows and m columns. Since H is surjective, the image of $\mathbf{A}$ is $\mathbf{S}$. Therefore $\operatorname{rank}(H) = \dim(\mathbf{S}) = m$, where $\dim(\mathbf{S})$ is the dimension of $\mathbf{S}$. The bit on the ith row and the jth column is 1 when address bit a_i is an input to the XOR-gate computing the jth set index bit. Consequently, the computation of the set index $\mathbf{s}$ can be expressed as the vector–matrix multiplication over $GF(2)$, where addition is computed as XOR and multiplication is computed as logical AND, denoted by $\mathbf{s} = \mathbf{a}H$.

2. Null spaces

Since matrices are considered as linear transformations, they can be characterized by their null spaces. The null space of a matrix H is the set of all addresses which map to index $\mathbf{0}$, namely $N(H) = \{\mathbf{x} : \mathbf{x}H = \mathbf{0}\}$.

The following theorem is well-known.

Theorem 2.2 (*See [15]*). *Let A be an $n \times m$ matrix. Then*

$$n = \operatorname{rank}(A) + \dim N(A).$$

Theorem 2.3. *Let A and B be $n \times m$ matrices. Then $N(A) \cap N(B) = N([A\ B])$, where $[A\ B]$ is the augmented matrix of A and B.*

Proof. The result is obtained by the following equivalence.

$$\mathbf{x} \in N(A) \cap N(B) \Leftrightarrow \mathbf{x}A = \mathbf{x}B = 0 \Leftrightarrow \mathbf{x} \in N([A\ B]). \quad \square$$

3. Design of efficient XOR-based hash functions

In this section we model efficient XOR-based hash functions when the number of address bits is $2m$, where m is even or odd. We propose the new model of XOR-based hash functions for two cases. We give XOR-based hash functions by the following.

Definition 3.1. We define XOR-based hash functions of four types as the following:

(i) $m = 2k - 1$ $(k \in \mathbf{N})$, where $\mathbf{N}$ is the set of all positive integers.

$$H_1 = \begin{pmatrix} T_1 \\ I_{2k-1} \end{pmatrix}, \qquad H_3 = \begin{pmatrix} T_3 \\ I_{2k-1} \end{pmatrix}, \qquad \text{where } I_{2k-1} \text{ is the identity matrix,}$$

$$T_1 = (t_{ij})_{(2k-1)\times(2k-1)}, \qquad t_{ij} = \begin{cases} 1, & \text{if } 1 \le i \le k, 1 \le j \le k-i+1, \\ 1, & \text{if } 1 \le i \le 2k-1,\ j = 2k-i, \\ 0, & \text{otherwise.} \end{cases}$$

$$T_3 = (t_{ij})_{(2k-1)\times(2k-1)}, \qquad t_{ij} = \begin{cases} 1, & \text{if } i = j = 1, \\ 1, & \text{if } 1 \le i \le 2k-1,\ j = 2k-i, \\ 1, & \text{if } k \le i \le 2k-1, 3k-i-1 \le j \le 2k-1, \\ 0, & \text{otherwise.} \end{cases}$$

(ii) $m = 2k$ ($k \in \mathbf{N}$)

$$H_2 = \begin{pmatrix} T_2 \\ I_{2k} \end{pmatrix}, \qquad H_4 = \begin{pmatrix} T_4 \\ I_{2k} \end{pmatrix}, \qquad \text{where } I_{2k} \text{ is the identity matrix,}$$

$$T_2 = (t_{ij})_{(2k)\times(2k)}, \quad t_{ij} = \begin{cases} 1, & \text{if } 1 \le i \le k, i \le j \le k, \\ 1, & \text{if } 1 \le i \le 2k-1,\ j = 2k-i+1, \\ 0, & \text{otherwise.} \end{cases}$$

$$T_4 = (t_{ij})_{(2k)\times(2k)}, \quad t_{ij} = \begin{cases} 1, & \text{if } k+1 \le i \le 2k, k+1 \le j \le i, \\ 1, & \text{if } 1 \le i \le 2k-1,\ j = 2k-i+1, \\ 0, & \text{otherwise.} \end{cases}$$

Example 3.2. The following are hash functions in Definition 3.1.

$$H_1 = \begin{pmatrix} 1&1&1&0&1 \\ 1&1&0&1&0 \\ 1&0&1&0&0 \\ 0&1&0&0&0 \\ 1&0&0&0&0 \\ 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{pmatrix}, \qquad H_3 = \begin{pmatrix} 1&0&0&0&1 \\ 0&0&0&1&0 \\ 0&0&1&0&1 \\ 0&1&0&1&1 \\ 1&0&1&1&1 \\ 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{pmatrix}$$

$$H_2 = \begin{pmatrix} 1&1&0&1 \\ 0&1&1&0 \\ 0&1&0&0 \\ 1&0&0&0 \\ 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix} \qquad H_4 = \begin{pmatrix} 0&0&0&1 \\ 0&0&1&0 \\ 0&1&1&0 \\ 1&0&1&1 \\ 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix}.$$

Theorem 3.3. (i) *All $m \times m$ matrices $T_i's$ in Definition 3.1 are nonsingular, where $i = 1, 2, 3, 4$.*
(ii) *Each $T_i \oplus I$ is nonsingular, where T_i is in Definition 3.1.*

Proof. By the following steps we can show that $|T_i| = 1$ and $|T_i \oplus I| = 1$ for $i = 1, 2, 3, 4$.

Step 1. For the cofactor expansion, use the row or column in which the number of $1's$ is only one.

Step 2. For the cofactor expansion, use the row such that the first element and the last element are 1 and the remaining elements are 0. And go to Step 1. □

		Column Index Matrix															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Row Index Matrix	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
	2	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
	3	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
	4	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
	5	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
	6	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
	7	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
	8	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
	9	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
	10	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
	11	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
	12	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
	13	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
	14	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	15	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8

Fig. 1. The mapping of elements to banks made by H_2.

Theorem 3.4. *Let* $H = \begin{pmatrix} T \\ I_m \end{pmatrix} = (\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_m}, \mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_m})^t$ *be the hash function in Definition 3.1, where* $T = (\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_m})^t$ *and* $I_m = (\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_m})^t$. *And let* $M_{ij} = (\mathbf{t_{m+1-i}}, \ldots, \mathbf{t_{m-1}}, \mathbf{t_m}, \mathbf{e_{m+1-j}}, \ldots, \mathbf{e_m})^t$, *where* $i + j = m$. *Here* A^t *is the transpose of* A. *Then* $\mathrm{rank}(M_{ij}) = m$.

Proof. (a) $j = 0$: In this case $M_{ij} = T$. Therefore $\mathrm{rank}(M_{ij}) = \mathrm{rank}(T) = m$ by Theorem 3.3.

(b) $j \geq 1$: Let $M'_{ij} = (\mathbf{t_m}, \mathbf{t_{m-1}}, \ldots, \mathbf{t_{m+1-i}}, \mathbf{e_{m+1-j}}, \ldots, \mathbf{e_m})^t$.

Case I: $T = T_1$ or $T = T_2$.

Since M'_{ij} is the lower diagonal matrix and the elements of the main diagonal of M'_{ij} are all 1 and thus $|M'_{ij}| = 1$. Hence the rows of M'_{ij} are independent. Since M_{ij} is row equivalent to M'_{ij}, $\mathrm{rank}(M_{ij}) = \mathrm{rank}(M'_{ij}) = m$.

Case II: $T = T_3$ or $T = T_4$.

Since M'_{ij} is the upper diagonal matrix and the elements of the main diagonal of M'_{ij} are all 1 and thus $|M'_{ij}| = 1$. Hence the rows of M'_{ij} are independent. Since M_{ij} is row equivalent to M'_{ij}, $\mathrm{rank}(M_{ij}) = \mathrm{rank}(M'_{ij}) = m$. □

By Theorems 3.3 and 3.4, the modeled XOR-based hash functions such as the functions defined in Definition 3.1 map the patterns (rows, columns, (anti)diagonal, and rectangles) without conflict. Fig. 1 shows the mapping of all elements to banks made by H_2 in Definition 3.1 when $m = 4$. So the elements belonging to one of the patterns are mapped to distinct banks. In Fig. 1, the row index is encoded in the upper 4 bits of the 8 bit vector and the column index is encoded in the lower 4 bits. For example, $8 = 1000$.

4. Conflict-free mapping in 2-way skewed-associative caches

In this section, we construct XOR-based hash functions for a 2-way skewed-associative cache of the same size that map the same patterns conflict-free.

1. Skewing on caches

Skewed-associative caches were proposed in [1–3]. Different hash functions are used for the distinct cache banks i.e., a line of data with address D may be mapped on line $H_1(D)$ in cache bank 1 or in $H_3(D)$ in cache bank 2, etc. This hardware modification incurs a very small hardware over cost when designing a new processor with on-chip caches since the hash functions can be chosen so that the implementation uses a very few gates. But a skewed-associative cache may help to increase the hit ratio of caches and then to increase the overall performance of a processor using a multibank cache structure [2,3].

In Fig. 2, A, B and C conflict for the same location in bank 1, but can be present at the same time, as they do not map to the same location in bank 2.

2. Interbank dispersion

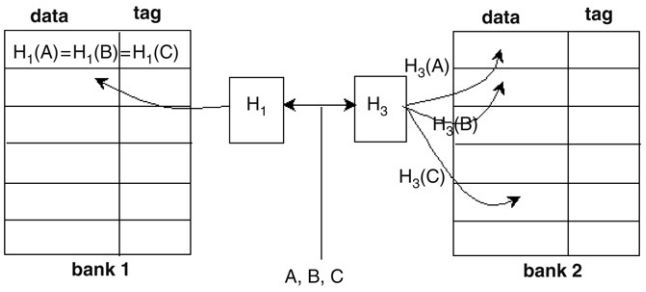

Fig. 2. Skewed-associative cache.

In a usual X-way set-associative cache, when $X + 1$ lines of data contend for the same set in the cache, they are all conflicting for the same place in the X cache banks: one of the lines must be rejected from the cache. Skewed-associative caches can avoid this situation by scattering the data: hash functions can be chosen such that whenever two lines of data conflict for a single location in cache bank i, they have very low probability to conflict for a location in cache bank j (Fig. 2). Ideally, hash functions may be chosen such as the set of lines that might be mapped on a cache line of bank i will be equally distributed over all the lines in the other cache banks. The number of similar columns determines the degree of interbank dispersion in a 2-way skewed-associative cache. The blocks that cannot be placed in one bank can be dispersed over all the sets in the other bank that compute the same value for the same columns. As hash functions have more columns in common, the possibility of dispersing blocks decreases. We define the degree of interbank dispersion between two hash functions as the following:

Definition 4.1. We define the degree of interbank dispersion (DID) between two XOR-based $2m \times m$ hash functions H_1, H_3 by

$$\mathrm{DID}(H_1, H_3) = \mathrm{rank}([H_1\ H_3]).$$

From the definition of the DID between two XOR-based hash functions H_1 and H_3, we can easily see that $0 \leq \mathrm{DID}(H_1, H_3) \leq 2m$. It is assumed that each bank has the same dimension. The hash functions of a 2-way skewed-associative cache should be designed such that the DID is maximal for every pair of hash functions.

Remark. Vandierendonck et al. [8] defined the DID between two hash functions H_1 and H_3 by using the concepts of supremum(infimum) of two functions and the concept of the dimension of column space. But we defined the DID by using only the concept of the rank of $[H_1\ H_3]$.

Example 4.2. (i) Let $m = 3$ and define two XOR-based hash functions H_1, H_3' as the following:

$$H_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad H_3' = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Then $\mathrm{rank}([H_1 H_3']) = 5$. So $\mathrm{DID}(H_1, H_3') = 5(< 6)$. Therefore $\dim N([H_1\ H_3']) = 1$ by Theorem 2.2. Since $N([H_1\ H_3']) = \{000000, 111101\} \neq \{000000\}$,

$$(000000)H_1 = (111101)H_1, (000000)H_3' = (111101)H_3'.$$

Thus H_1 and H_3' cannot avoid conflict.

(ii) Consider two XOR-based hash functions H_1 and H_3 defined in Definition 3.1.

		Value of H_1							
		0	1	2	3	4	5	6	7
Value of H_3	0	0:0	5:2	4:5	1:7	7:1	2:3	3:4	6:6
	1	5:3	0:1	1:6	4:4	2:2	7:0	6:7	3:5
	2	4:7	1:5	0:2	5:0	3:6	6:4	7:3	2:1
	3	1:4	4:6	5:1	0:3	6:5	3:7	2:0	7:2
	4	7:5	2:7	3:0	6:2	0:4	5:6	4:1	1:3
	5	2:6	7:4	6:3	3:1	5:7	0:5	1:2	4:0
	6	3:2	6:0	7:7	2:5	4:3	1:1	0:6	5:4
	7	6:1	3:3	2:4	7:6	1:0	4:2	5:5	0:7

Fig. 3. Illustration of H_1 and H_3 and interbank dispersion.

$$H_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad H_3 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Then $\text{rank}([H_1\ H_3]) = 6$, and thus $\text{DID}(H_1, H_3) = 6$. Therefore $\dim N([H_1\ H_3]) = 0$ by Theorem 2.2. Since $N([H_1\ H_3]) = \{000000\}$, $(000000)H_1 = (111101)H_1$ but $(000000)H_3 = (000) \neq (100) = (111101)H_3$.

Every address in the main memory is mapped to a set in bank 1 by H_1 and to a set in bank 2 by H_3. These functions are illustrated in a two-dimensional plot (Fig. 3). Each axis is labeled with the possible set indices for that bank. Every address is displayed in the grid in a position that corresponds to its set indices in each bank. For two vectors x and y in the same row, $xH_3 = yH_3$ but $xH_1 \neq yH_1$. Similarly, for two vectors u and v in the same column, $uH_1 = vH_1$ but $uH_3 \neq vH_3$.

Therefore this skewed-associative cache can avoid conflict by H_1 and H_3.

Fig. 3 shows that the DID of H_1 and H_3 is maximal. Also the element (5 : 3) represents (101 011).

The following theorem characterizes the maximality of the DID of H_1 and H_3.

Theorem 4.3. *The degree of interbank dispersion between H_1 and H_3 is maximal if and only if $N([H_1\ H_3]) = \{\mathbf{0}\}$.*

Proof. By Theorem 2.2 and Definition 4.1, the proof is completed by the following:

$$\text{DID}([H_1 H_3]) = 2m \Leftrightarrow \text{rank}([H_1 H_3]) = 2m$$
$$\Leftrightarrow \dim N([H_1 H_3]) = 0 \Leftrightarrow N([H_1 H_3]) = \{\mathbf{0}\}. \quad \square$$

Lemma 4.4. *Let T_1 and T_3 be matrices in Definition 3.1, where $m = 2k - 1 (k \geq 3)$. Then* $\text{rank}([T_1 \oplus T_3]) = m$.

Proof. By the row operation and column operation of matrix $T_1 \oplus T_3$, we get an equivalent matrix

$$\begin{pmatrix} I_{k-1} & A \\ O & I_k \end{pmatrix},$$

where A is a $(k-1) \times k$ nonzero matrix. Hence

$$\text{rank}(T_1 \oplus T_3) = \text{rank}([I_{k-1}\ A]) + \text{rank}([O\ I_k]) = (k-1) + k = 2k - 1 = m. \quad \square$$

The following theorem shows that the DID between the proposed hash functions H_1 and H_3 is maximal. We can see that these functions map $2m$ bit address to m bit set index without conflict for a 2-way skewed-associative cache.

Theorem 4.5. *Let $H_1 = \begin{pmatrix} T_1 \\ I_m \end{pmatrix}$ and $H_3 = \begin{pmatrix} T_3 \\ I_m \end{pmatrix}$ for T_1 and T_3 be matrices in Definition 3.1, where $m = 2k - 1$ $(k \geq 3)$. Then* $\text{rank}([H_1\ H_3]) = 2m$.

Proof. Since $[H_1 \ H_3] = \begin{pmatrix} T_1 & T_3 \\ I_m & I_m \end{pmatrix} \Rightarrow \begin{pmatrix} I_m & I_m \\ O & T_1 \oplus T_3 \end{pmatrix} \Rightarrow \begin{pmatrix} I_m & O \\ O & T_1 \oplus T_3 \end{pmatrix}$, by Lemma 4.4, $\operatorname{rank}([H_1 \ H_3]) = \operatorname{rank}([I_m \ O]) + \operatorname{rank}([O \ T_1 \oplus T_3]) = m + m = 2m$. □

Corollary 4.6. *Let* $H_1 = \begin{pmatrix} T_1 \\ I_m \end{pmatrix}$ *and* $H_3 = \begin{pmatrix} T_3 \\ I_m \end{pmatrix}$ *for* T_1 *and* T_3 *be matrices in Definition 3.1, where* $m = 2k-1$ $(k \geq 3)$. *Then* $N([H_1 \ H_3]) = \{\mathbf{0}\}$.

Remark. By the similar proof, we can show that Theorem 4.5 and Corollary 4.6 hold for H_2 and H_4, where $m = 2k$.

5. Conclusion

In this paper we designed new XOR-based hash functions by using the concepts of rank and null space. We constructed conflict-free hash functions which are of different types according to whether m is even or odd. Also we showed that the DID between the proposed hash functions is maximal. By the results we showed that these functions map without conflict for a 2-way skewed-associative cache.

References

[1] A. Seznec, A new case for skewed-associativity, Technical Report PI-1114, IRISH, 1997.
[2] F. Bodin, A. Seznec, Skewed-associativity improves program performance and enhances predictability, IEEE Trans. Comput. 46 (1997) 530–544.
[3] A. Seznec, A case for two-way skewed associative caches, in: Proc. 20th Ann. Int'l Symp. Computer Architecture, 1993, pp. 169–178.
[4] B.R. Rau, Pseudo-randomly interleaved memory, in: Proc. 18th Ann. Int'l Symp. Computer Architecture, 1991, pp. 74–83.
[5] M. Kharbutli, K. Irwin, Y. Solihin, J. Lee, Using prime numbers for cache indexing to eliminate conflict misses, in: Proc. 10th Int'l Symp. High Performance Computer Architecture, 2004, pp. 288–299.
[6] A.J. Smith, Cache memories, ACM Comput. Surv. 14 (1982) 473–530.
[7] J.M. Frailong, W. Jalby, J. Lenfant, XOR-schemes: A flexible data organization in parallel memories, in: Proc. 1985 Int'l Conf. Parallel Processing, 1985, pp. 276–283.
[8] Hans Vandierendonck, Koen De Bosschere, XOR-based hash functions, IEEE Trans. Comput. 54 (2005) 800–812.
[9] Hans Vandierendonck, Koen De Bosschere, Efficient profile-based evaluation of randomising set index functions for cache memories, in: International Symposium on Performance Analysis of Systems and Software, 2001, pp. 120–127.
[10] Hans Vandierendonck, Koen De Bosschere, Evaluation of the performance of polynomial set index functions, in: Proc. Workshop Duplicating, Deconstructing, and Debunking, 2002, pp. 31–41.
[11] S.J. Cho, U.S. Choi, Y.H. Hwang, Y.S. Pyo, H.D. Kim, S.H. Heo, Computing phase shifts of maximum-length 90/150 cellular automata sequences, in: Proc. ACRI 2004, in: LNCS, vol. 3305, 2004, pp. 31–39.
[12] S.J. Cho, U.S. Choi, H.D. Kim, Analysis of complemented CA derived from a linear TPMACA, Comput. Math. Appl. 45 (2003) 689–698.
[13] S.J. Cho, U.S. Choi, H.D. Kim, Behavior of complemented CA whose complement vector is acyclic in a linear TPMACA, Math. Comput. Modelling 36 (2002) 979–986.
[14] S.J. Cho, U.S. Choi, Y.H. Hwang, H.D. Kim, Analysis of complemented CA derived from linear hybrid group CA, Comput. Math. Appl. 53 (2007) 54–63.
[15] S.J. Leon, Linear Algebra, 6th ed., Prentice-Hall, Inc., 2002.